# INTEGRATING SOFTWARE STACKS

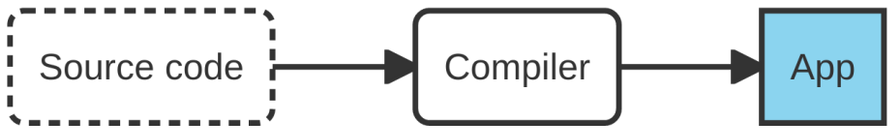
## With **BuildStream 2.0** and the **Remote Execution API**

Sam Thursfield

OSSEU 2022

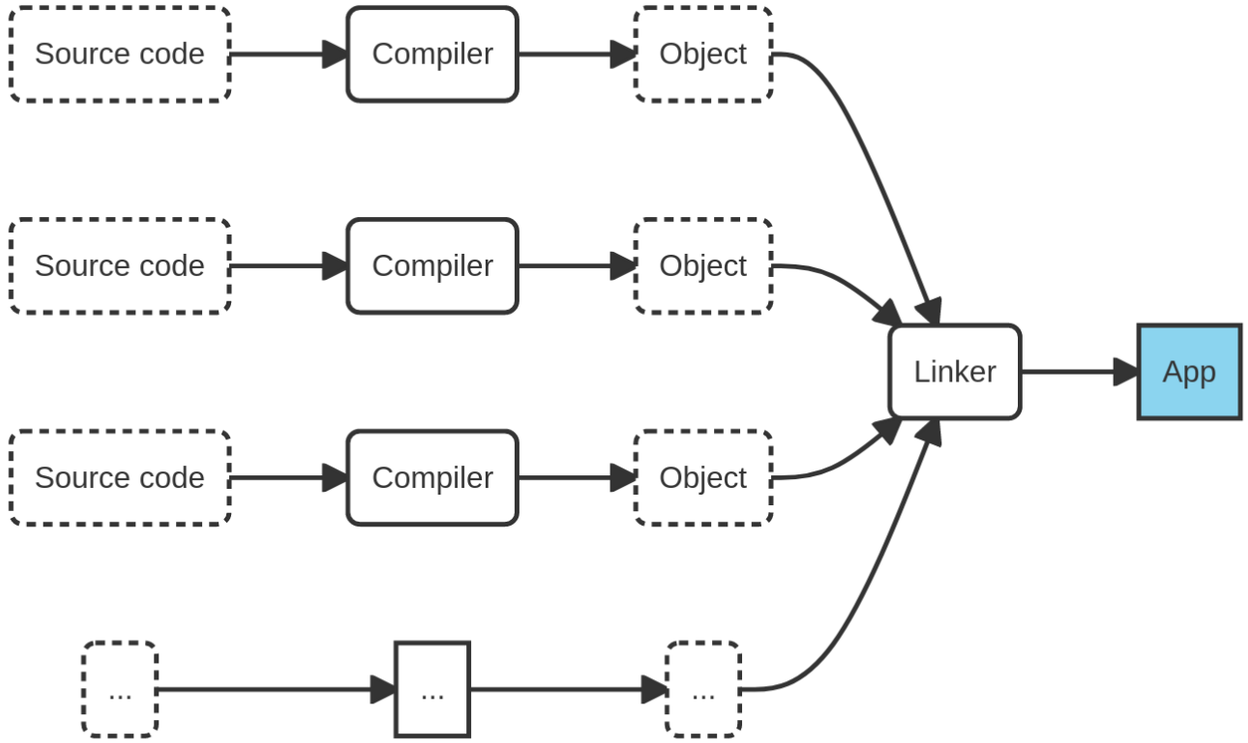
# THE BASIC PROBLEM

Compiling source code to binaries



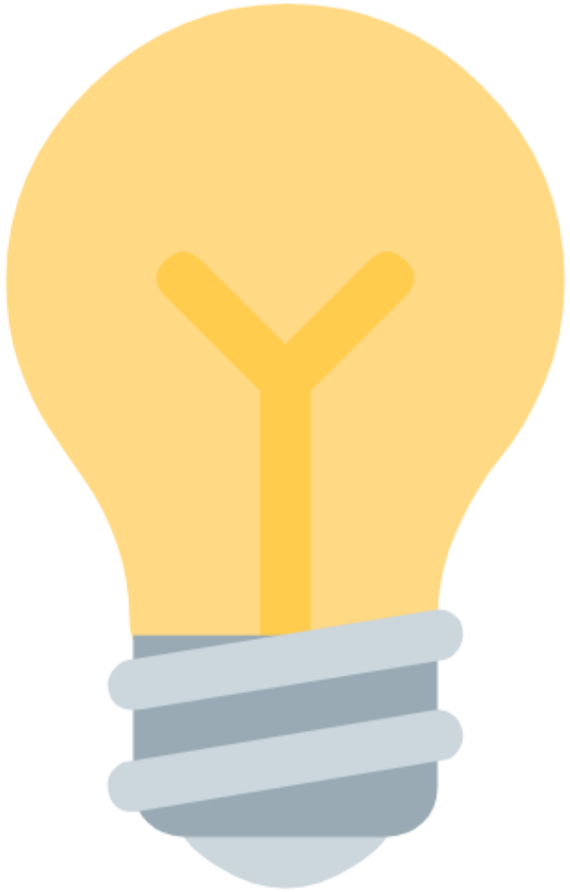
# THE BASIC PROBLEM

Compiling source code to binaries



# A BRIEF HISTORY OF INTEGRATION TOOLS

In the beginning was: **Make**



# A BRIEF HISTORY OF INTEGRATION TOOLS

In the beginning was: **Make**

| | **Compiler** | **Make** |
|---|---|---|
| Dependency tracking | None | File level |

# A BRIEF HISTORY OF INTEGRATION TOOLS

In the beginning was: **Make**

| | **Compiler** | **Make** |
|---|---|---|
| Dependency tracking | None | File level |
| Multiple repos | No | No |
| Repeatable builds | No | No |

# A BRIEF HISTORY OF INTEGRATION TOOLS

Then came the **packaging tools**...

| | **Compiler** | **Make** | **Packaging tools** |
|---|---|---|---|
| Dependency tracking | None | File-level | Package-level |
| Multiple repos | No | No | Yes |
| Repeatable builds | No | No | Mostly |

# A BRIEF HISTORY OF INTEGRATION TOOLS

- **Meta-build tools** (Buildroot, BuildStream, Yocto)
- **Containers** (Docker, Podman, Flatpak)
- **Continuous integration**
- and more...

| | **Compiler** | **Build tools** | **Integration tools** |
|---|---|---|---|
| Dependency tracking | None | File-level | Package-level |
| Multiple repos | No | No | Yes |
| Repeatable builds | No | No | Mostly |

# INTEGRATING AT SCALE IN 2022

One tool to "rule them all" ...?

# INTEGRATING AT SCALE IN 2022

> *Make each program do one thing well.*

— Doug McIlroy, The Bell System Technical Journal, July-August 1978

# INTEGRATING AT SCALE IN 2022

> *Make each program do one thing well.*

— Doug McIlroy, The Bell System Technical Journal, July-August 1978

## How can our integration tools work together?

# THE REMOTE EXECUTION API

Introduced in 2017, by team at Google working on Bazel.

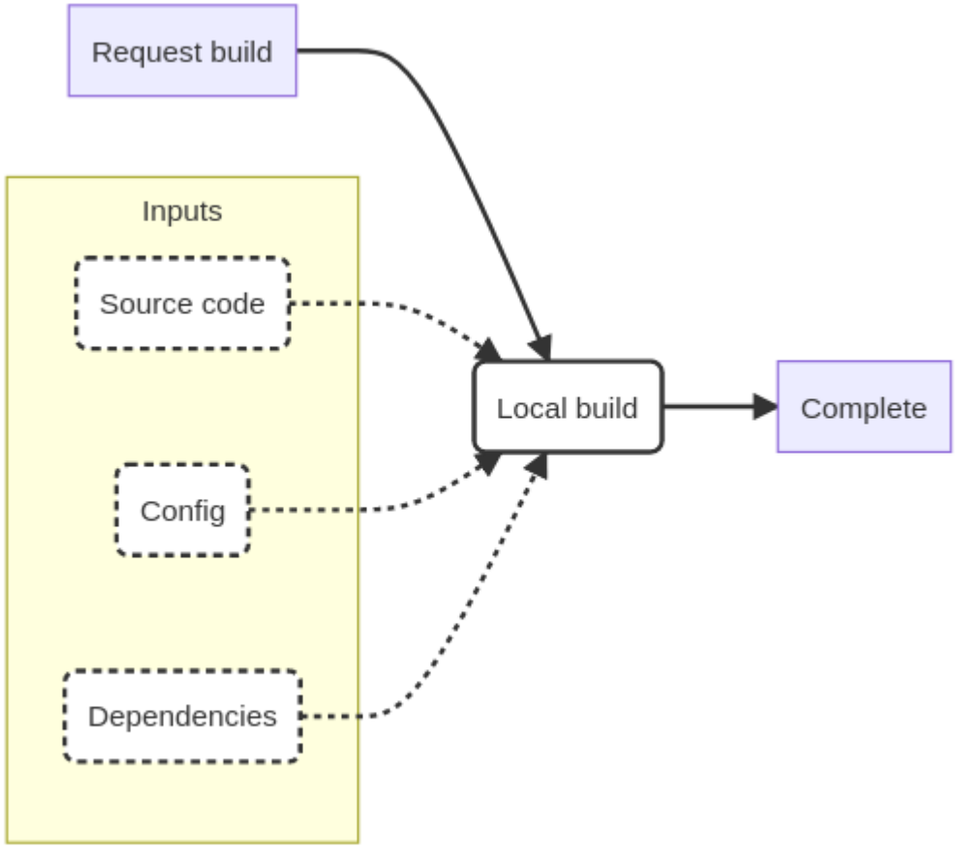
Designed around:

- Content addressable storage for directories and files
- An execution request interface
- A **result cache**

Building blocks for any integration pipeline you can think of.

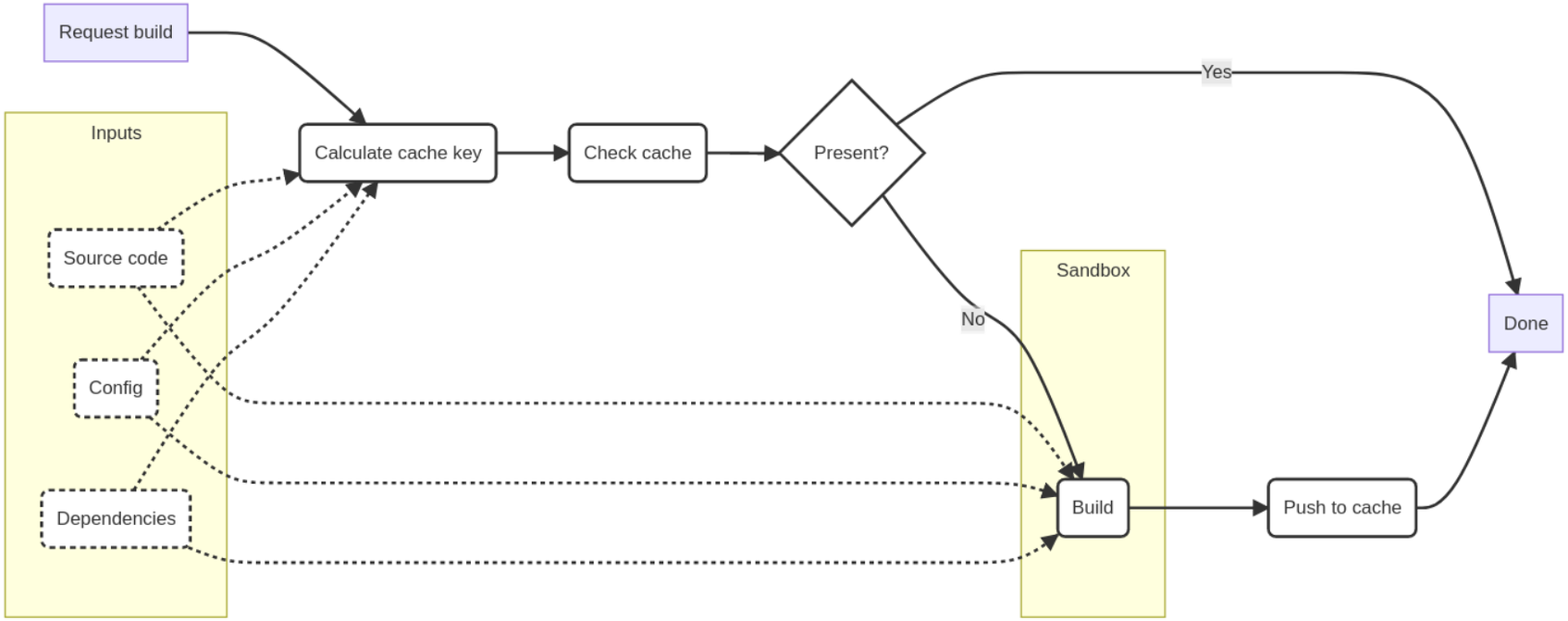
# THE REMOTE EXECUTION API

## Traditional build tools



# THE REMOTE EXECUTION API

## Content-addressed caching




For cache to be trusted and shared, cache key must capture all factors that can affect build output.

The cache and the builder can be local, or remote.

# THE REMOTE EXECUTION API

## New Remote Execution API Draft

Subscribe

146 views

**Alexis Hunt**
to bazel-...@googlegroups.com

18 Apr 2017, 18:20:06

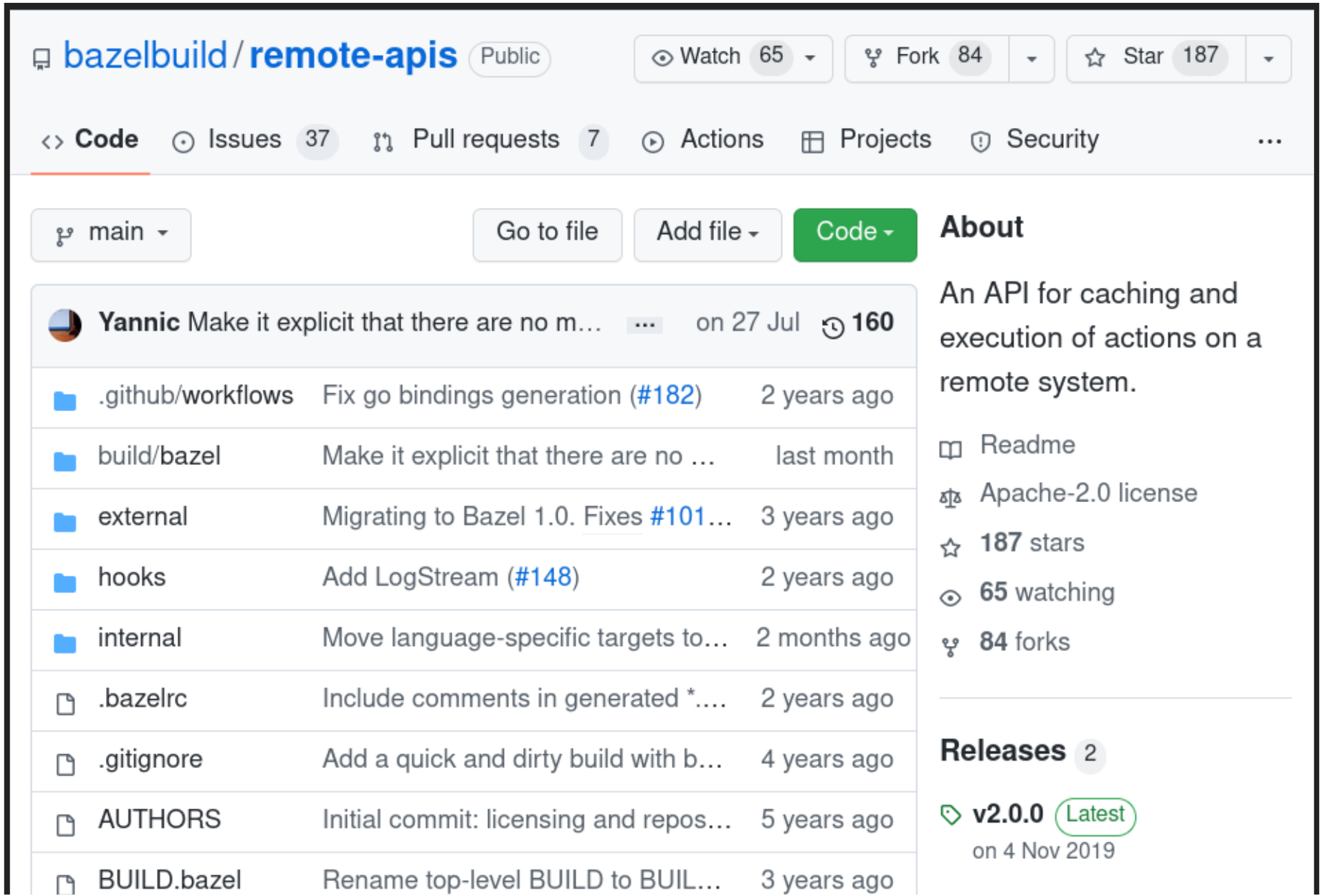
Hello Bazel community!

A few months ago, we circulated a draft API for remote execution & caching in Bazel. This was implemented as a prototype, and based on our work with this, we've prepared a revised draft of the API. The basic functionality is still the same:
- A content-addressable storage for storing directory trees and files.
- An execution request interface.
- A result cache.

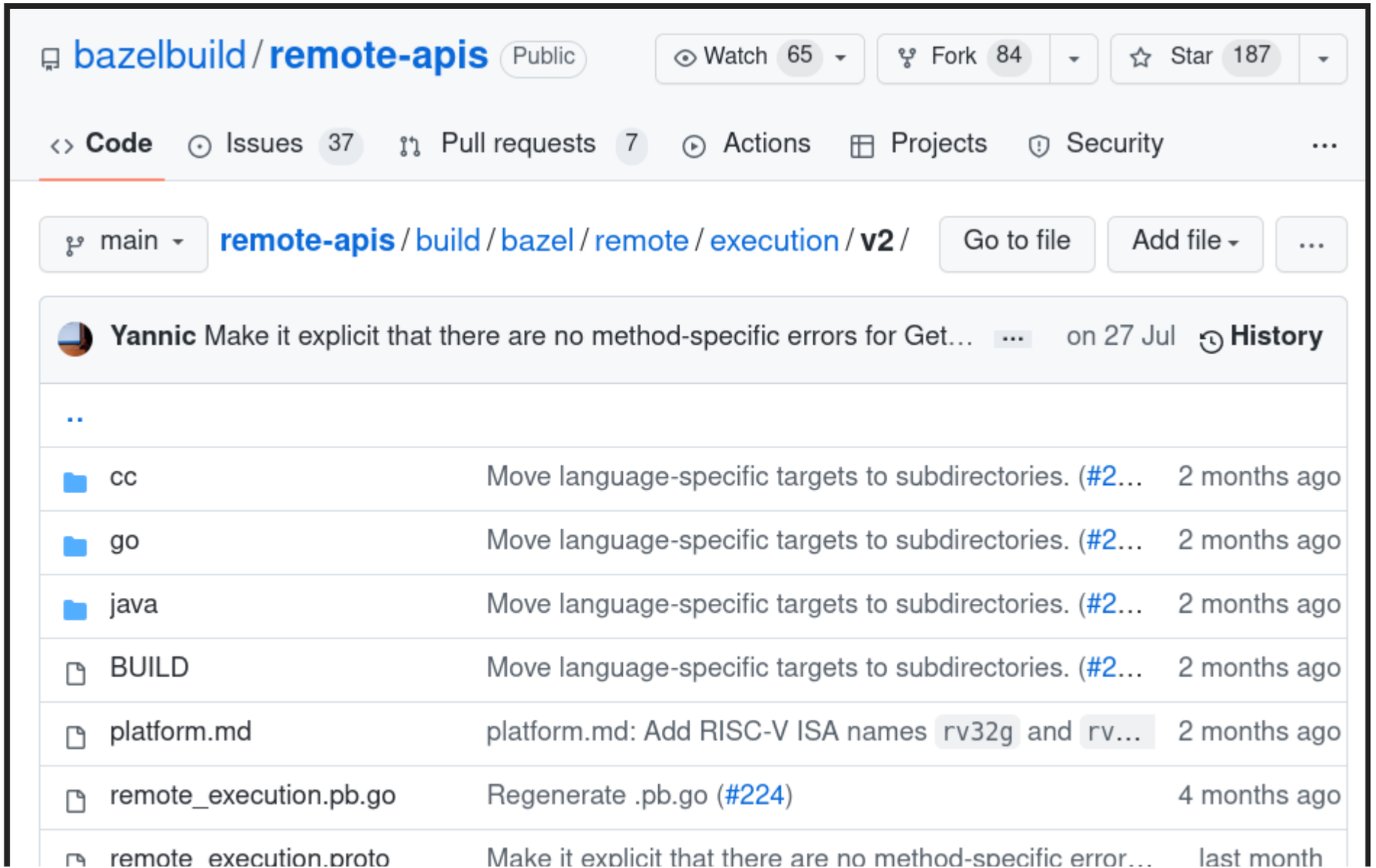
Many of the details have been changed based on feedback and where we thought shortcomings in the original design were. We've also endeavoured to bring the API more in line with Google API standards.

The design document (including the API spec) is available at https://goo.gl/TB49ED. Please offer any feedback or suggestions you have in the document itself. Once there is consensus on the API, we will work to update the prototype in Bazel to match.

# THE REMOTE EXECUTION API



# THE REMOTE EXECUTION API



# REAPI CLIENTS

**README.md**

There are a number of clients and services using these APIs, they are listed below.

## Clients

These tools use the Remote Execution API to distribute builds to workers.

- Bazel
- BuildStream
- Goma Server
- Pants
- Please
- Recc

# REAPI CLIENTS

All these tools support remote caching and distributed builds via the Remote Execution API.

| | Name | Dependency tracking | Repeatable builds | Multiple repos |
|---|---|---|---|---|
| Compiler tools | Goma Server | - | - | - |
| | Recc | - | - | - |
| Build tools |  Bazel | File-level | Partly[1] | Partly |
| |  Pants | File-level | Partly[1] | No |
| |  Please | File-level | Yes | No |
| Integration tools |  BuildStream | Element-level | Yes | Yes |

# REAPI CLIENTS

All these tools support remote caching and distributed builds via the Remote Execution API.

| | Name | Dependency tracking | Repeatable builds | Multiple repos |
|---|---|---|---|---|
| Compiler tools | Goma Server | - | - | - |
| | Recc | - | - | - |
| Build tools |  Bazel | File-level | Partly[1] | Partly |
| |  Pants | File-level | Partly[1] | No |
| |  Please | File-level | Yes | No |
| Integration tools |  BuildStream | Element-level | Yes | Yes |

Compiler tools: replacements for ccache + distcc.

# REAPI CLIENTS

All these tools support remote caching and distributed builds via the Remote Execution API.

| | Name | Dependency tracking | Repeatable builds | Multiple repos |
|---|---|---|---|---|
| Compiler tools | Goma Server | - | - | - |
| | Recc | - | - | - |
| Build tools |  Bazel | File-level | Partly[1] | Partly |
| |  Pants | File-level | Partly[1] | No |
| |  Please | File-level | Yes | No |
| Integration tools |  BuildStream | Element-level | Yes | Yes |

Compiler tools: replacements for ccache + distcc.

Build tools: replacements for Make

1. Bazel and Pants sandboxing doesn't hide host environment in any way.

# REAPI CLIENTS

All these tools support remote caching and distributed builds via the Remote Execution API.

| | Name | Dependency tracking | Repeatable builds | Multiple repos |
|---|---|---|---|---|
| Compiler tools | Goma Server | - | - | - |
| | Recc | - | - | - |
| Build tools |  Bazel | File-level | Partly[1] | Partly |
| |  Pants | File-level | Partly[1] | No |
| |  Please | File-level | Yes | No |
| Integration tools |  BuildStream | Element-level | Yes | Yes |

Compiler tools: replacements for ccache + distcc.

Build tools: replacements for Make

1. Bazel and Pants sandboxing doesn't hide host environment in any way.

Integration tools: designed for integrating multiple components, such as packages.

# THINGS TO BE AWARE OF

- Bazel isn't magic
  - Adapting an existing project to Bazel means rewriting its entire build system
  - Requires a BUILD file for each 3rd party dependency
  - Sandbox can be leaky.
  - Hard to extend beyond built-in languages, no plugin mechanism
  - Hard to land changes upstream
  - CLI is not pleasant to use

- BuildStream isn't magic
  - Designed more for integrators and less for developers
  - If making frequent changes to slow-to-build elements, be prepared to wait
  - Sandboxing helps ensure repeatable builds, but nothing is foolproof

# REAPI INFRASTRUCTURE

| Name | Language | Storage | Execution |
|---|---|---|---|
| Bazel Buildfarm | Java | disk, S3, shard, ... | local |
| BuildBarn | Go | disk, S3, shard | local |
| BuildGrid | Python/C++ | disk, Redis, S3, shard, ... | local, bwrap, OCI, ... |
| Please-servers | Go | Google Cloud Services | local |
| Scoot | Go | - | local |

I recommend BuildBarn or BuildGrid for use with BuildStream

# THINGS TO BE AWARE OF

- Protobufs might make you sad
  - Pin "known-good" versions to avoid segfaults
  - Public code, private development
- Infrastructure experts required
  - BuildBarn, Buildfarm and BuildGrid all require some work to deploy
- Cache expiry
  - For best results, use the cache as a cache

# REAPI WISHLIST

## Clients:

- More build tools supporting REAPI
  - sccache (link)
  - Distro packaging tools
- Improve BuildStream experience for developers
  - Faster rebuilds, perhaps with Recc inside Bst

## Infrastructure:

- Wider support
  - Artifactory supporting REAPI caching (issue)
- Easier deployment and management
- Better documentation

**Find out more about the REAPI: https://github.com/bazelbuild/remote-apis**

# BUILDSTREAM

- Integration tool comparable to Buildroot or BitBake
- Open source, Apache Foundation project
- Designed around strong caching
- Control all inputs to build process and sandbox build process
- Version 2.0 (releasing this month) designed around REAPI

# BUILDSTREAM - EXAMPLE PROJECT

elements/base.bst

```
kind: import
description: |

    Alpine Linux base runtime

sources:
- kind: tar

  # This is a post doctored, trimmed down system image
  # of the Alpine linux distribution.
  #
  url: alpine:integration-tests-base.v1.x86_64.tar.xz
  ref: 3eb559250ba82b64a68d86d0636a6b127aa5f6d25d3601a79f79214dc9703639
```

elements/hello.bst

```
kind: autotools
description: |

    Hello world example from automake

variables:

  # The hello world example lives in the doc/amhello folder.
  #
  # Set the %{command-subdir} variable to that location
  # and just have the autotools element run its commands there.
  #
  command-subdir: doc/amhello

sources:
- kind: tar
  url: gnu:automake-1.16.tar.gz
  ref: 80da43bb5665596ee389e6d8b64b4f122ea4b92a685b1dbd813cd1f0e0c2d83f

depends:
- base.bst
```

# BUILDSTREAM - EXAMPLE PROJECT

elements/base.bst

```
kind: import
description: |

    Alpine Linux base runtime

sources:
- kind: tar

  # This is a post doctored, trimmed down system image
  # of the Alpine linux distribution.
  #
  url: alpine:integration-tests-base.v1.x86_64.tar.xz
  ref: 3eb559250ba82b64a68d86d0636a6b127aa5f6d25d3601a79f79214dc9703639
```

elements/hello.bst

```
kind: autotools
description: |

    Hello world example from automake

variables:

  # The hello world example lives in the doc/amhello folder.
  #
  # Set the %{command-subdir} variable to that location
  # and just have the autotools element run its commands there.
  #
  command-subdir: doc/amhello

sources:
- kind: tar
  url: gnu:automake-1.16.tar.gz
  ref: 80da43bb5665596ee389e6d8b64b4f122ea4b92a685b1dbd813cd1f0e0c2d83f

depends:
- base.bst
```

# BUILDSTREAM - EXAMPLE PROJECT

elements/base.bst

```
kind: import
description: |

    Alpine Linux base runtime

sources:
- kind: tar

  # This is a post doctored, trimmed down system image
  # of the Alpine linux distribution.
  #
  url: alpine:integration-tests-base.v1.x86_64.tar.xz
  ref: 3eb559250ba82b64a68d86d0636a6b127aa5f6d25d3601a79f79214dc9703639
```

elements/hello.bst

```
kind: autotools
description: |

    Hello world example from automake

variables:

  # The hello world example lives in the doc/amhello folder.
  #
  # Set the %{command-subdir} variable to that location
  # and just have the autotools element run its commands there.
  #
  command-subdir: doc/amhello

sources:
- kind: tar
  url: gnu:automake-1.16.tar.gz
  ref: 80da43bb5665596ee389e6d8b64b4f122ea4b92a685b1dbd813cd1f0e0c2d83f

depends:
- base.bst
```

# BUILDSTREAM - EXAMPLE PROJECT

elements/base.bst

```
kind: import
description: |

    Alpine Linux base runtime

sources:
- kind: tar

  # This is a post doctored, trimmed down system image
  # of the Alpine linux distribution.
  #
  url: alpine:integration-tests-base.v1.x86_64.tar.xz
  ref: 3eb559250ba82b64a68d86d0636a6b127aa5f6d25d3601a79f79214dc9703639
```

elements/hello.bst

```
kind: autotools
description: |

    Hello world example from automake

variables:

  # The hello world example lives in the doc/amhello folder.
  #
  # Set the %{command-subdir} variable to that location
  # and just have the autotools element run its commands there.
  #
  command-subdir: doc/amhello

sources:
- kind: tar
  url: gnu:automake-1.16.tar.gz
  ref: 80da43bb5665596ee389e6d8b64b4f122ea4b92a685b1dbd813cd1f0e0c2d83f

depends:
- base.bst
```

# BUILDSTREAM - EXAMPLE PROJECT

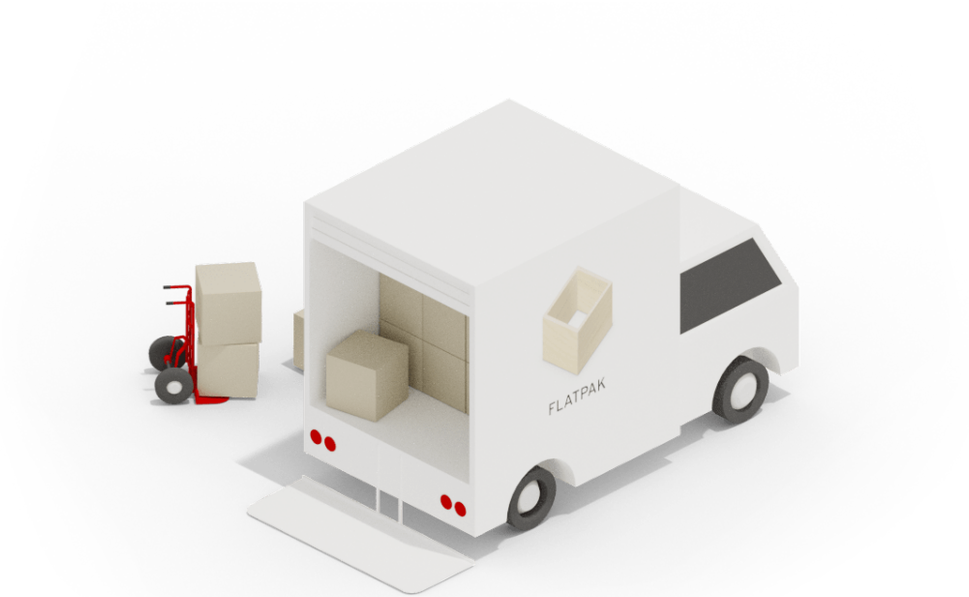
## project.conf

```
# Unique project name
name: autotools

# Minimum required BuildStream version
min-version: 2.0

# Subdirectory where elements are stored
element-path: elements

# Define some aliases for the tarballs we download
aliases:
  alpine: https://bst-integration-test-images.ams3.cdn.digitaloceanspaces.com/
  gnu: http://ftpmirror.gnu.org/gnu/automake/

plugins:
- origin: pip
  package-name: buildstream-plugins
  elements:
  - autotools
```

https://github.com/apache/buildstream/tree/master/doc/examples/autotools

# CASE STUDY 1

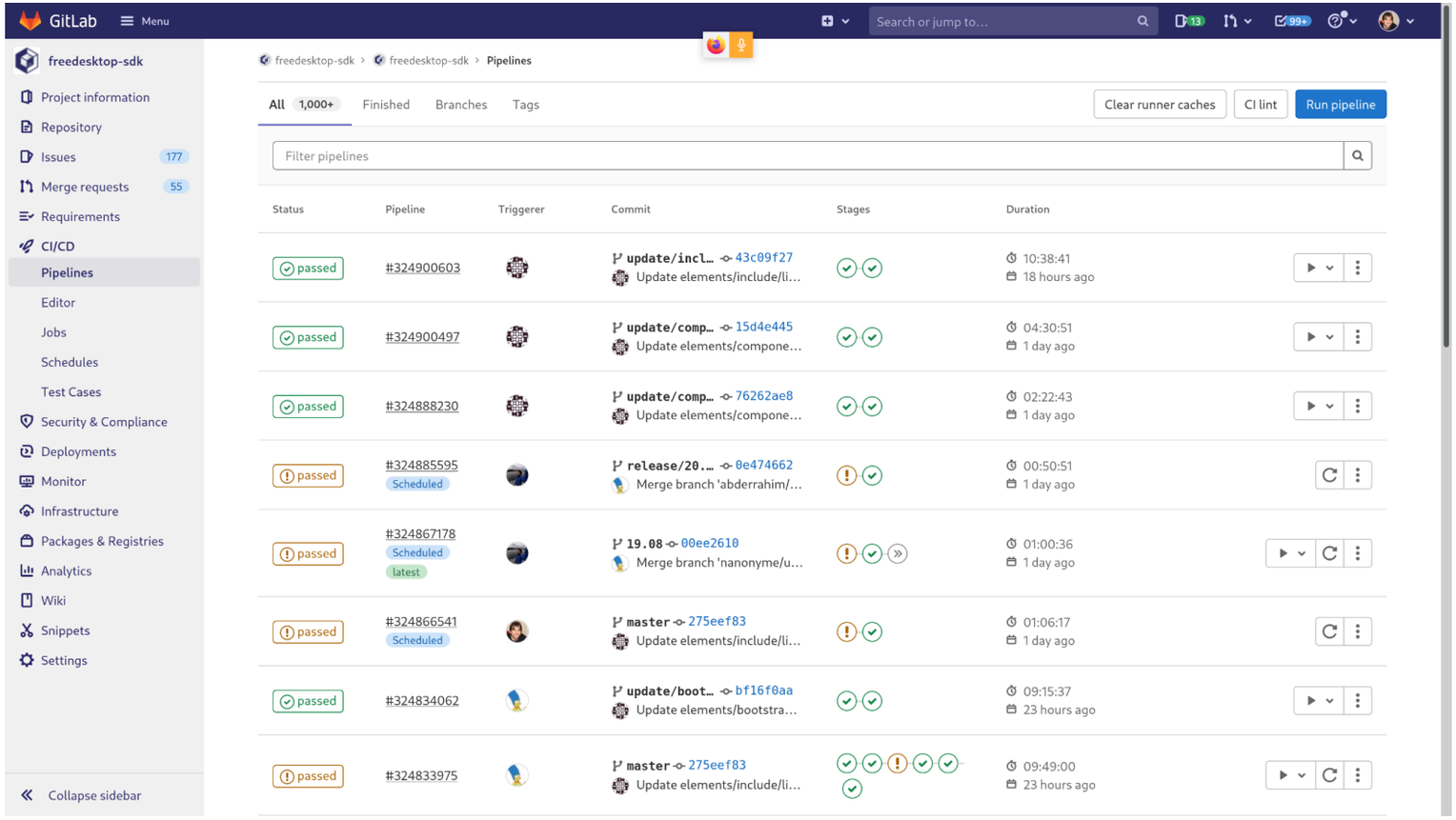
## Freedesktop SDK and Runtime




freedesktop-sdk

# CASE STUDY: FREEDESKTOP SDK

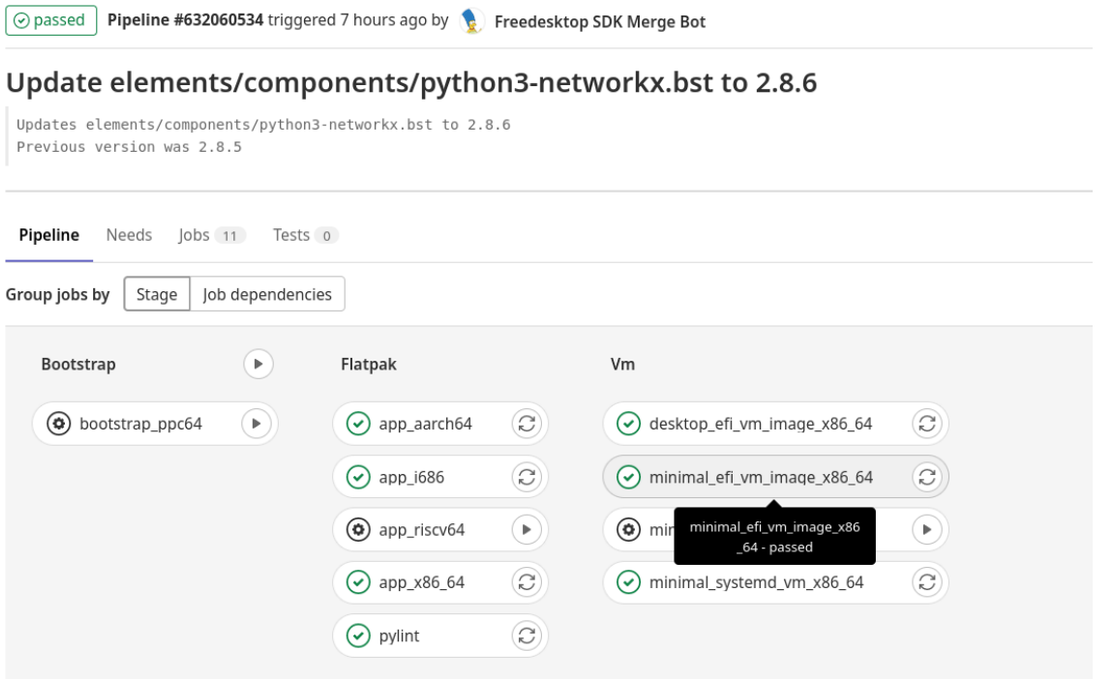
- Stable base runtime for most Flatpak apps.
- Integrates > 600 open source components across 4 architectures with BuildStream.
- Largely volunteer-powered.
- Driven by Gitlab CI



# CASE STUDY: FREEDESKTOP SDK

## Benefit from using BuildStream and REAPI:

- **Shared cache server** provided by BuildBox (BuildGrid)
- CI pushes all built elements to shared cache
- "Full rebuild" is almost never necessary
- Component updates automated
- Downstream projects benefit from upstream cache



# CASE STUDY 2

## Building for safety with DCS

# CASE STUDY: DCS

DCS = "Deterministic Construction Service"

Qualified tool integration for building complex software (e.g. Linux-based operating systems) for safety-critical systems

A design pattern for safely building software.

# CASE STUDY: DCS

DCS = "Deterministic Construction Service"

Qualified tool integration for building complex software (e.g. Linux-based operating systems) for safety-critical systems

A design pattern for safely building software.

Repeatable builds are safe builds:

- Re-run of DCS process reproduces exactly the same binary fileset
- Reproducibility is shown to be independent of the specific instantiation of DCS

# CASE STUDY: DCS

## DCS = "Deterministic Construction Service"

Qualified tool integration for building complex software (e.g. Linux-based operating systems) for safety-critical systems

A design pattern for safely building software.

Repeatable builds are safe builds:

- Re-run of DCS process reproduces exactly the same binary fileset
- Reproducibility is shown to be independent of the specific instantiation of DCS

Allows us to:

- Upgrade or modify tools used to construct safety-critical software and verify that these changes have no impact on the output binaries
- Perform fine-grained impact analysis on the constructed software when updating source and dependencies

# CASE STUDY: DCS

## DCS = "Deterministic Construction Service"

Qualified tool integration for building complex software (e.g. Linux-based operating systems) for safety-critical systems

A design pattern for safely building software.

Repeatable builds are safe builds:

- Re-run of DCS process reproduces exactly the same binary fileset
- Reproducibility is shown to be independent of the specific instantiation of DCS

Allows us to:

- Upgrade or modify tools used to construct safety-critical software and verify that these changes have no impact on the output binaries
- Perform fine-grained impact analysis on the constructed software when updating source and dependencies

Reference implementation uses BuildStream.

# CASE STUDY: DCS

Qualified using ISO 26262 safety standard.

exida

The manufacturer may use the mark:

exida • CERTIFIED • FUNCTIONAL SAFETY

Revision 1.2 October 5, 2021
Surveillance Audit Due
October 1, 2024

ANAB
ANSI National Accreditation Board
ACCREDITED
ISO/IEC 17065
PRODUCT CERTIFICATION BODY
#1004

Certificate / Certificat
Zertifikat / 合格証

COD 1912002 C001

*exida* hereby confirms that the:

**Deterministic Construction Service Reference Implementation (DCS)**
**Codethink Ltd.**
**Manchester - UK**

Has been assessed per the relevant requirement of:
**ISO 26262: 2018**
and meets requirements providing a level of integrity to:
**ASIL D Qualified Tool**

**Tool Functions:**
DCS is used to deterministically initiate and direct software construction such that the construction of a particular version can be reproduced exactly.

**Application Restrictions:**
The tool must be used in accordance with the procedures and constraints documented in the Safety Manual.

exida Certification

Evaluating Assessor

Certifying Assessor

Page 1 of 2

More information about safe software:

- see Paul Albertella speak on **Friday, 11.45**
- get involved in ELISA

# HAPPY INTEGRATING

Please invest in your build + integration pipelines, and...

Check out **BuildStream 2.0** at https://buildstream.build

Sam Thursfield

OSSEU 2022

CodeThink